This is the pre-peer reviewed version of the following article: Mihali, R., Sobh, T. and Vamoser, D. (2004), SKED: A course scheduling and advising software. Comput. Appl. Eng. Educ., 12: 1–19. doi: 10.1002/cae.10054, which has been published in final form at http://onlinelibrary.wiley.com/doi/10.1002/cae.10054/abstract.

SKED: A Course Scheduling and Advising Software

RAUL MIHALI, TAREK SOBH and DAMIR VAMOSER

Department of Computer Science and Engineering, University of Bridgeport,

Bridgeport, CT 06601, USA

May 2000

SKED: A Course Scheduling and Advising Software

RAUL MIHALI, TAREK SOBH and DAMIR VAMOSER
Department of Computer Science and Engineering, University of Bridgeport, Bridgeport, CT 06601, USA
May 2000

Abstract:

Each year at the beginning of a new academic semester, most advisors face a very common and particularly tedious and time consuming problem: deciding for each student what course schedule would be ideal for the following semester so that the student would graduate in the fastest possible time and also have his / her specific preferences and pre-requisites satisfied.

The factors that have to be considered vary from school specific requirements such as course pre-requisites, co-requisites, spring and fall offerings, to student specific ones, such as transferred credits or the subjective desire to choose or not a given choice. While some advisors might be able to derive reasonable solutions in reasonable amount of time, the process takes most of the advising time. The student will have to "trust" the advisor that the given schedule is the best choice, and in many cases the results will later on prove that the student could have actually graduated faster, or that specific school requirements have been violated or simply that the student's load and preference could have been balanced better.

This paper presents a software application that can completely solve the presented problem. Once the school specific data and requirements have been set, for any specific student information, the application will search and output the schedules that will allow the student to graduate in the fastest number of semesters / quarters possible. Depending on the factors and data considered, the execution time varies from few seconds to few minutes. Currently, we have successfully tested and implemented the application at the University of Bridgeport, CT, USA.

1. Introduction

Post secondary education is usually being categorized in fields of studies defined as *majors*. Each *major* has its unique class curriculum and requirements, usually preset for years and undergoing limited infrastructure changes. Since usually a student can choose one or very few *majors* to study, the problem is considered at *the major* level.

The completion of a *major* usually implies that a student goes through a given number of courses, following department and inter-department requirements, spring/fall restrictions, maximal number of credits per semester as well as any particular requirements that may apply to him/her as a result of an advisor suggestion. Most of the *majors* would typically require around eight semesters for completion and depending on the number of credits taken at a time the student would be considered *freshman, sophomore, junior or senior*. The courses that are to be taken are mainly directly relevant to the *major*, while others are general requirements for all the *majors*, or particular *pre* or *co* requisites for various relevant courses.

A *pre-requisite* of a course *A* is defined as a course that a student needs to have taken already in order to be able to take course *A*. A course can have none or many *pre-requisites* and all of them need to be satisfied.

A *co-requisite* of a course *A* is a defined as a course that a student needs to have taken in order to be able to take course *A*, but can also be taken in the same semester with course *A*.

An academic year is composed of a *fall semester* and a *spring semester* (or a 3-4 quarters in a quarter based system - please note that the application can be easily adjusted to suit custom academic schedules). While usually most of the general requirement courses are offered in both semesters, *major* specific or particular courses are often offered once a year.

The *maximum number of credits per semester* is the number of credits that limit the total credits that a student can take during any given semester.

The *maximum number of semesters* represents the maximum number of semesters in which a student should try to graduate.

As an example, as of now, the Bachelors of Computer Science degree at University of Bridgeport, requires the completion of 8 semesters at an average load of 18 credits per semester and a total of 131 credits. Most of the courses are directly related to the Computer Science field and general requirements consist of courses of math, physics, English composition, etc. (for a clearer view, please note that through the paper we will elaborate on the above example)

2. The Algorithm

The goal of the algorithm is to provide the course schedules that would allow a student to graduate in the fastest possible time, from any semester that he/she currently might be in. The *major* specific information described in section 1 is being used as data and guidance rules for the search process, together with various student dependent information [1].

The idea of an exhaustive search is not really a suitable solution. In the cases we have tested, it resulted in searching times ranging from few seconds to few days [2]. To overcome this search time problem, we have formulated and implemented a goal-seeking algorithm tailored to our specific problem (similar algorithms can be found in [3]).

Each course is being given a *requirement cost*. The *requirement cost of a course* is being defined as the longest possible chain of *pre requisites* that contains the respective course. For example, if course *D* has as pre requisite course *C*, and course *C* has as pre requisite course *B*, and course *B* has as prerequisite course

A, this would make a *chain of pre requisites* of *requirement-cost* 3 for course *A*. The longest chain that can be found for course *A* will be its associated *requirement-cost*. To reflect a worst case scenario, for this cost, the co-requisites are being treated as pre-requisites.

Based on the requirement cost, the algorithm will try to schedule the courses with the highest cost first, thus minimizing the number of semesters a student needs to take [4].

A course is also being associated an *availability cost*. The *availability cost* of a course is the number of semesters that one has to go through before one would be able to take that course. The cost depends on pre requisites, co requisites and spring/fall offerings. For example, a course with an availability cost of 3 can be taken 3 semesters away from now, this as a result of its co and pre requisites combined with the spring/fall offerings. A course with the *availability cost 0* reflects a course for which all the pre and co requirements have been satisfied and the course is also offered in the current semester (Fall or Spring).

Having defined the above two costs, a general scheme of the algorithm can be formulated:

1) If the student has already taken (transferred) any courses, update the co and pre requisites, as well as the list of *to be taken* courses.
2) For all the *to be taken* courses, calculate the *availability cost*.
3) From all the *to be taken* courses with the *availability cost* 0, calculate the *requirement cost*.
4) From all the *to be taken* courses with the *availability cost* 0, pick up those that have the highest *requirement cost*, until the *maximum number of credits per semester* has not been exceeded. Let us call this a *closed list* of courses [5].
5) If there was a *closed list* of courses, then if the lowest *requirement cost* in this list coincides with the highest *requirement cost* of the rest of the courses selected at 3, remove ("put back") all courses with this cost from the *closed list.*
6) From the courses with the *availability cost* 0 that are not in the *closed list* and have the highest *requirement cost*, form combinations [6] and keep only those that when added with the *closet list* credits do not exceed the *maximum number of credits per semester*. Let us call the results *open lists*, representing lists of possible semesters.
7) For each of the lists from the *open lists*, repeat from step 1) until all the courses have been successfully scheduled, and record for future display the schedules with the quickest completion time.

Example:

Based on the taken courses and the semester for which the algorithm is scheduling, the following courses prove to have an *availability cost* of zero: AD101, CPE286, CPE471, ENGL204, HUMC202, MATH214, MATH301, MATH314, ME223, SSCC201 (please see Appendix for a description of the courses, these being part of a set of courses on which it will be worked throughout the paper). Basically these would be all the courses that a student could theoretically attend the following semester. The courses, sorted descending by the maximum *requirement cost*, have the following information (Table1):

Course	Credits	Cost
ME223	3	4
CPE286	3	3
ENGL204	1	2
MATH301	3	2
SSCC201	3	2
AD101	3	1
HUMC202	3	1
MATH314	3	1
CPE471	3	0
MATH214	3	0

Table 1.

Having the maximum number of credits per semester set to 18, the algorithm will pick up for the *close list* the following courses ME223, CPE286, ENGL204, MATH301, SSCC201, AD101 (see step 4). According to step 5, the *closed list* will omit course AD101 and remain ME223, CPE286, ENGL204, MATH301 and SSCC201.

Based on step 6, the following three *open lists* will be created:
ME223, CPE286, ENGL204, MATH301, SSCC201, AD101;
ME223, CPE286, ENGL204, MATH301, SSCC201, HUMC202;
ME223, CPE286, ENGL204, MATH301, SSCC201, MATH314;
Each of them will be recursively explored further, as step 7 indicates.

3. The Software Package

In its current stage, the software package has been developed using Microsoft Visual Basic and is composed of 4 distinct parts. The *Data Manager* part, allows for the managing of the necessary data and rules that mainly pertain to the *major* as a whole and that typically do not need to be modified for each student. The *Profiler* allows for the managing of student specific information that changes from student to student. The *Schedules* is the part where the results of the algorithm will be output, and *Others* is a part that contains various global settings, as well as a mini web server mode that allows application to be used over a web browser. By having this structure, various personnel can modify and work with specific information. For example the registrar would normally use the *Data Manager* to add / remove / edit courses. The advisors would use the *Profiler* to adjust student specific information, while the students would use the *Schedules* to select their desired schedule of study.

3.1. The Data Manager

Figure 1. The Data Manager

The Data Manager tab (Figure 1) was designed to facilitate the input and managing of all the data and rules (course requirements, school preferences etc) that would pertain specifically to a major, and would normally not need adjustment over short periods of time. The idea is to have this data loaded and verified one time, and then used as a shared database by the advisers of a certain *major*.
For more convenient handling of the information, the data managing has been divided into 5 different options:
Courses, Requirements, Spring/Fall Offerings, Special Courses and Special Requirements.

Courses

The *courses* window allow for the direct input, editing or removal of the course specific information (Figure 2).

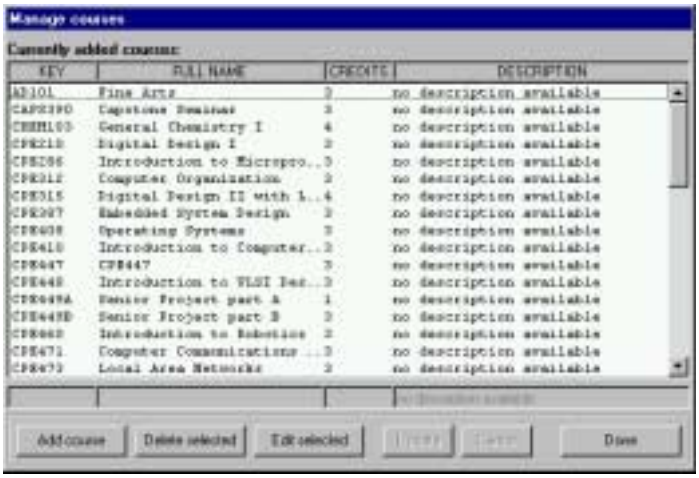

Figure 2. The course managing window

For a course, the software will store a unique KEY, made as a combination of letters and digits and used internally throughout the algorithm functions when referring the courses. CREDITS, represents the number of credits of a course, numeric value that is used internally. FULL NAME and DESCRIPTION are simply informative fields and have significance solely for the user. All the necessary courses should be added here. The courses visible in Figure 2 and the following figures are part of the courses that are needed at University of Bridgeport for the Bachelor of Computer Engineering.

Requirements

The *requirements* window allows for the managing of the requirements between classes (Figure 3).

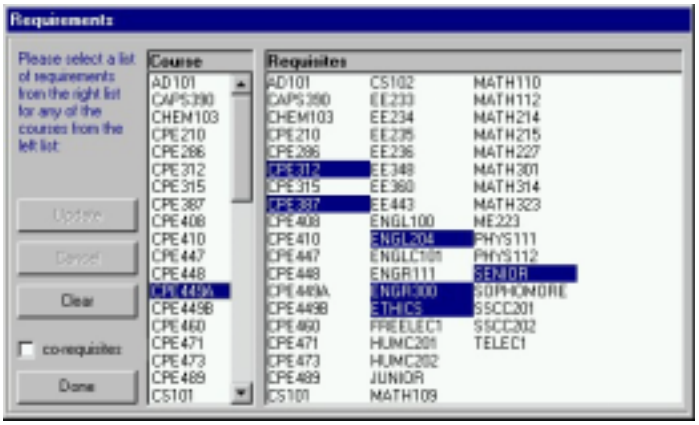

Figure 3. Requirements

The left most list of courses contains all the courses introduced through the *Courses* option. For each course, various requisite courses can be chosen. Note on the right list, courses such as FRESHMAN, SOPHOMORE, JUNIOR and SENIOR. These courses are added by default by the software and only have the role of adding better control on course requirements, in fact counting for zero credits. By checking the *co-requisites* check box, the requisites in the right most list will behave as co-requisites for the selected course on the left.
Also note that the software will check for redundant or circular reference requisites and not allow them. A *redundant requisite* appears when a pre-requisite of a course A has as pre-requisite that coincides with another pre-requisite of the course *A*.
For example, if CPE449A requires CPE387 and CPE449B requires CPE449A, it would be redundant to have CPE449B require CPE387 as well and the software will detect and notify of any such case.
A *circular reference* appears when a requisite for a course happens to have that course as a requisite as well, directly or indirectly. For example, if CPE449B requires CPE449A and CPE449A requires CPE387, there would be a circular reference if CPE387 would require CPE449B. The software will detect and warn accordingly of such problems.

Fall / Spring Offerings

The Fall / Spring Offerings window allow to select which course is being offered in which semester (Figure 4). Courses from the left most list can be selected and added to any of the two right lists representing the fall or the spring semesters.

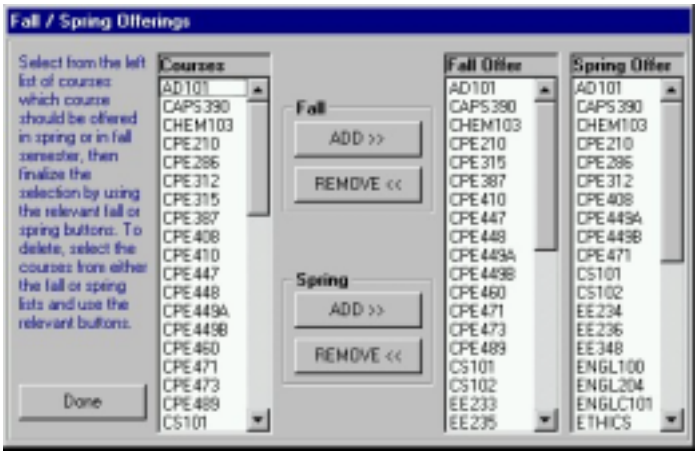

Figure 4. Fall / Spring Offerings

For example, the course AD101 is being offered both fall and spring semesters, while ENGL204 is only offered in Spring. The software will warn if any of the courses are not offered at all.

Special Courses / Groups

In many cases, it can happen that out of a group of various courses only a few need to be taken, whichever the student chooses. For example, out of CPE410, CPE460, CPE471 and CPE473, only two courses need to be taken, whichever the student prefer (Figure 5).

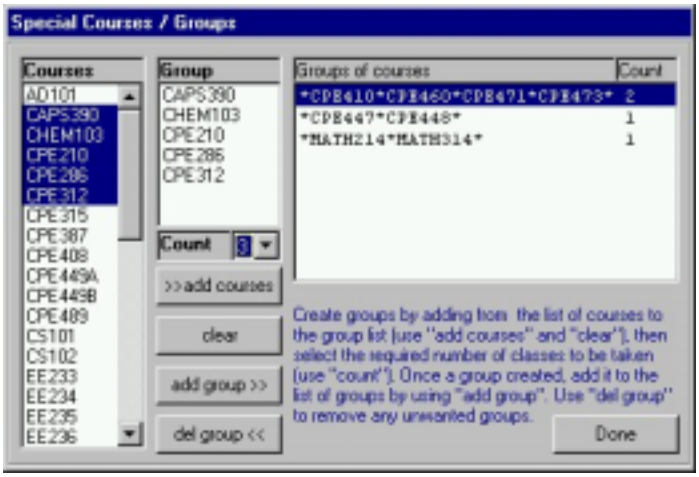

Figure 5. Special Courses / Groups

Through this window, such groups of classes can be specified. From the list of all the courses (left most), the desired courses need to be added to the middle list by using the *>>add courses* command, and once the desired number to be taken has been chosen from *Count*, the group can be added. Note that each course has its own requisites and from the way they are selected, this can facilitate or not a faster completion of the *major*. The algorithm takes this fact into account when searching.

Special Requirements

The Special Requirements window was added as a means of "enforcing" a student to take a certain class no later than a certain semester, as to provide a better control for advising (Figure 6).

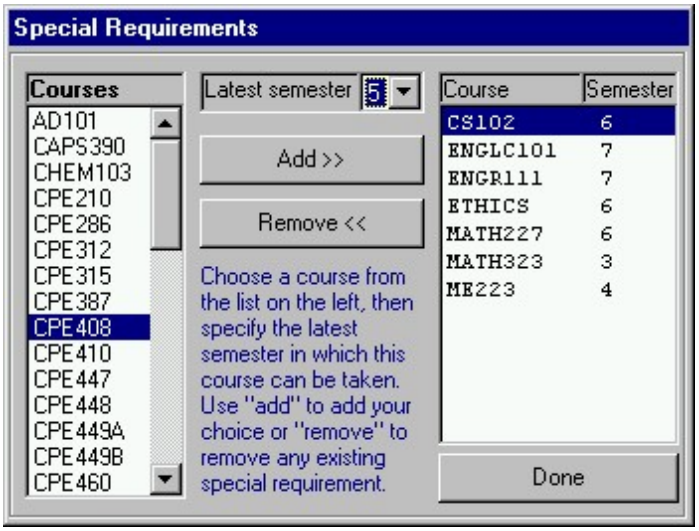

Figure 6. Special Requirements

For example, MATH227 does not have a high *requirement cost*, and normally the algorithm will try to place it in a later semester, first dealing with the "urgent" courses. This fact might not be appropriate when the course in case might be an easy one and should not be left for junior or senior years, despite that it does not have an explicit chain of pre / co requisites to fill.
Through the special Requirements window, a user can control such issues. By forcing a *requirement cost* of 6 for MATH227, this will oblige the algorithm to place this course at least 6 semesters before the completion of the degree, or in the case the degree of the student has less than 6 semesters remaining to completion, to assign it immediately in the first semester if possible.
From the left most list of course, the user would choose a course, assign it a cost through the *latest semester* value and add it.

3.2. The Profiler

The profiler allows to control the information that is specific to a student, such as the courses that he / she has taken, or that the advisor might want to adjust on a case to case basis to obtain better results (Figure 7).

Figure 7. The Profiler

The profiler is structured as follows:
Taken Courses list, allows the advisor to select the course that the student has taken so far.
Status Definition, allows the advisor to adjust the total number of credits that would define any of the *sophomore, junior or senior statuses.* While normally they would be defined at once for all the students, it proves to be convenient when trying to enforce various advisor preferences. In the example from Figure 7, if the student has taken more than 37 credits and less than 71, he / she has a SOPHOMORE status. Anything less than 37 credits is considered FRESHMAN, and anything more than 105 credits is considered SENIOR.
The *Maximum number of semesters, Maximum number of credits per semester and Starting semester* are being defined in the profiler as well.
The advisor also has the choice to ignore some of the restrictions, such as *Special Courses, Special Requirements or fall / spring restrictions.* Such options prove to be powerful especially when advising junior/senior students who would go for independent studies instead of the normal courses or have a difficult schedule that would allow exceptions from the department.
Ignore substitutes display has just a formatting result in the output solutions. When the option is not checked, the courses that are members of a *Special Group* will be displayed as a group, allowing for a more compact view (see Section 4 for details).
The *Search Criteria* allows changing the searching algorithm being used. The *Full Heuristic (recommended)* option, or the default one, will use the search algorithm as described in this paper, and should be the only one needed. For testing purposes, for evaluating speed difference and performance, there is the *Light Heuristic* search choice and the *Exhaustive search* one. The *Light Heuristic* method is very similar to the full heuristic (default) choice, except that it does not expand the *closed list* to various *open lists*. Note that this can often result in no solutions for the problem. The *Exhaustive Search* algorithm, will not create a *Closed list* based on *requirement costs*, instead it will apply full combinations on the list of course that are available to be taken at a certain time. Note that this option can be extremely slow and does not assure optimal solutions. Again, the search choice option has only been implemented for testing and debugging purposes. The same applies for the *Show Costs* option, which will display the *requirement cost* for all the courses.
A profile can also be saved or loaded, thus maintaining easily the records of each student for future reference.

3.3. The Schedules

Once the profiler has been adjusted for a particular scenario, and preferentially saved, by switching to the *Schedules* tab and clicking *go* the software will start searching for the optimal course schedules (Figure 8). A progress report is displayed and the process can be cancelled at any time if the solutions already found are sufficient.

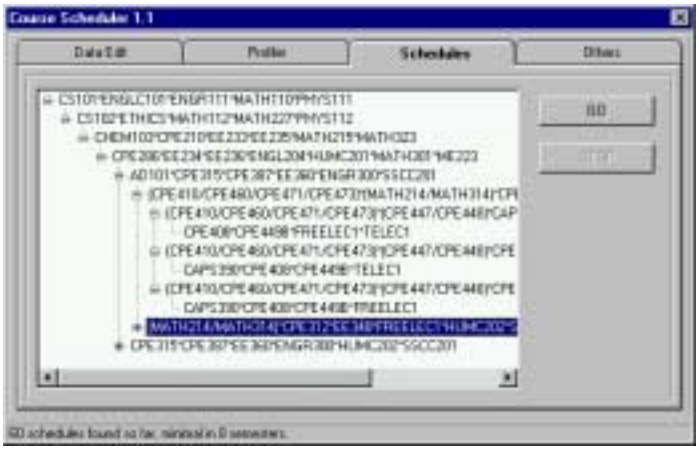

Figure 8. The Schedules

The solutions found are being displayed as a tree, each leaf representing the set of courses for a semester. For example CS101*ENGLC101*ENGR111*MATH110*PHYS111 is the only optimal choice for a first semester, while the fifth semester can be either AD101*CPE315*CPE387*EE360*ENGR300*SSCC201 or CPE315*CPE387*EE360*ENGR300*HUMC202*SSCC201. Once one of the two semester has been chosen, the semesters from its sub-tree should be considered for continuation.
The *Special Groups* of courses defined through the *Data Manager*, are normally being displayed in parentheses, to note the fact that any of the courses from the parentheses enclosed set can be chosen. For example, (CPE410/CPE460/CPE471/CPE473) would suggest to the student to choose any of the four courses and only one of them. Multiple such parentheses can occur through a semester, case in which the student should choose accordingly for each of them. If the *Ignore substitutes* display in the Profiler is being checked, these groups of courses will be expanded in multiple solutions with single choices.
A text file with all the solutions is being output as well.

3.4. Other Features

In its current form, the application can also be run as an HTTP server, allowing access to the profiler through a web browser, thus a more flexible way of sharing the database among advisors. A minimal level of security is implemented as well, allowing for IP filtering and user/password based access. Future work and directions will be detailed in the next section.5

4. A Complete Example. Design and Implementation History

By presenting a complete example on which the application was fully tested and built, this section should provide a good understanding of the current design of the application, what limitations and problems have been encountered and how were they corrected.

KEY	NAME	CREDITS
AD101	Fine Arts	3
CAPS390	Capstone Seminar	3
CHEM103	General Chemistry I	4
CPE210	Digital Design I	3
CPE286	Introduction to Microprocessors	3
CPE312	Computer Organization	3
CPE315	Digital Design II with Laboratory	4
CPE387	Embedded System Design	3
CPE408	Operating Systems	3
CPE410	Introduction to Computer Architecture	3
CPE447	FPGA Design	3
CPE448	Introduction to VLSI Design	3
CPE449A	Senior Project part A	1
CPE449B	Senior Project part B	3
CPE460	Introduction to Robotics	3
CPE471	Computer Comm. I: System Analysis	3
CPE473	Local Area Networks	3
CPE489	Software Engineering	3
CS101	Introduction to Computing I	3
CS102	Introduction to Computing II	3
EE233	Network Analysis I	3
EE234	Network Analysis II	2
EE235	Network Analysis I Lab	1
EE236	Network Analysis II Lab	1
EE348	Electronic Circuits I	3
EE360	Controls	3
EE443	Applied Digital Signal Processing	3
ENGL100	Basic Composition	3
ENGL204	Technical Writing for Comp. Sci. & Eng.	1
ENGLC101	Composition and Rhetoric I	3
ENGR111	Introduction to Engineering I	3
ENGR300	Economics and Management of Eng.	1
ETHICS	Integrated Studies In Comp (INSTC101)	3
FREELEC1	Free Elective One	3
HUMC201	Introduction to Humanities I	3
HUMC202	Introduction to Humanities II	3
MATH109	Precalculus Mathematics	4
MATH110	Calculus and Analytic Geometry I	4
MATH112	Calculus and Analytic Geometry II	4
MATH214	Linear Algebra	3
MATH215	Calculus and Analytic Geometry III	4
MATH227	Discrete Structures	3
MATH301	Differential Equations	3
MATH314	Numerical Methods	3
MATH323	Probability and Statistics	3
ME223	Materials Science for Engineers	3
PHYS111	Principles of Physics I	4
PHYS112	Principles of Physics II	4
SSCC201	Introduction to the Social Sciences I	3
SSCC202	Introduction to the Social Sciences II	3
TELEC1	Technical Elective 1	3

Table 2. List of courses for the Computer Engineering Major

At University of Bridgeport, a student that has just been admitted as an undergraduate freshman in Computer Engineering field, will have to complete a total of 131 credits, through a schedule of 8 semesters at an average of 18-19 credits per semester. The courses that are to be taken are highlighted in Table 2, where the **key** is the short form of the name, commonly used when referring to courses.

For the given scenario, there are three groups of courses out of which only a few need to be taken: out of CPE 410, CPE471, CPE473 and CPE460, only two (any) need to be taken; out of MATH214 and MATH314, only one (any), and also only one out of CPE447 and CPE448.

Also note that MATH109 and ENGL100 can be usually replaced by *placement exams*, thus bringing the total to 131 credits.

The following courses (given in alphabetical order here) are being offered in the *fall semester*:

AD101, CAPS390, CHEM103, CPE210, CPE315, CPE387, CPE410, CPE447, CPE448, CPE449A, CPE449B, CPE460, CPE471, CPE473, CPE489, CS101, CS102, EE233, EE235, EE360, EE443, ENGL100, ENGLC101, ENGR111, ENGR300, FREELEC1, HUMC201, HUMC202, MATH109, MATH110, MATH112, MATH215, MATH227, MATH323, PHYS111, PHYS112, SSCC201, SSCC202, TELEC1

And the following are offered in the *spring semester*:

AD101, CAPS390, CHEM103, CPE210, CPE286, CPE312, CPE408, CPE449A, CPE449B, CPE471, CS101, CS102, EE234, EE236, EE348, ENGL100, ENGL204, ENGLC101, ETHICS, FREELEC1, HUMC201, HUMC202, MATH109, MATH110, MATH112, MATH214, MATH215, MATH227, MATH301, MATH314, ME223, PHYS111, PHYS112, SSCC201, SSCC202, TELEC1.

The following courses are considered *core requirements*: CHEM103, CPE210, CPE286, CS101, EE233/235, ENGR111, ENGR300, MATH215, MATH301, MATH323, ME223.

The following courses are considered *program requirements:* CPE312, CPE315, CPE387, CPE408, CPE447/448, CPE449, CPE489, CS102, CS227, EE234, EE348, EE360, EE443, MATH214/314.

Note that most of the courses that are being offered both semesters are core requirements and courses that are usually general requirements for many majors (i.e. ENGLC101, ENGL100, MATH109).

The final and most important constraint exists in the set of *requisites and pre-requisites* that exists between courses, shown in figures 15 and 16. Figures 11, 12, 13, 14 and 15 show respectively the Design Sequence, the Software Sequence, the Integrated Software / Hardware Design Sequence, the Hardware Sequence and the Electrical Engineering Sequence of courses. Based on the presented data, the university came up with a suggested course schedule shown in figure 22. The schedule has been designed manually, by student advisors / professors. While the schedule certainly meets the requirements imposed for the Computer Engineering degree, any change or customization for a student's needs (especially the case of a transfer student) would be questionable.

The first goal of the application was to find a suitable, fairly normalized and scalable data structure that could contain the given information. While a trivial Microsoft Access database seemed a sufficient start in the beginning, after few months of testing and debugging we have reached the currently presented *Data Manager*. It is essential to have various filters that can guarantee the integrity and quality of the data input by a user.

As a next step, we designed quickly a brute force (combinatorial) algorithm, mainly as an immediate way of exercising the versatility of our data structure and to get an idea regarding the execution time (see Section 3.2, *Search Options*).

Some of the problems appeared already: unacceptable execution time (a first solution was output after more than 24 hours of execution time); performing the various data manipulation routines directly on the Access table was a significant slowdown as well.

At this stage, we started to implement the suggested algorithm. The skeleton idea was primarily derived from the sequence of requisites and pre-requisites that suggest a certain "order of importance" for courses.

The data has been copied into memory and all the data manipulation routines were simplified and changed to work completely from the memory.

For a faster implementation and result, the co-requisites have been considered pre-requisites and the groups of special courses have been ignored.

The execution time has been reduced significantly and the algorithm started to promising outputs.

However, due to the incomplete implementation and consideration of the problem, the application was not outputting completely realistic solutions. We then adjusted the algorithm to be able to work with groups of courses (note that while a student has the choice to choose which to take, each has its own list of requirements and some choices could improve the overall output). Co-requisites have also been added and handled properly. The concepts of SOPHOMORE, JUNIOR and SENIOR have also been implemented.

The new results were more promising and closer to viable solutions, however, it became obvious that many of the courses happen to have very few or no pre-requisites and also a low requirement cost, a fact that would make the algorithm consider them primarily for the later semesters.

The following is an example of such a problem:

CHEM103,CPE210,CS101,MATH110,PHYS111
MATH314,CS102,ENGLC101,MATH112,PHYS112
CPE315,EE233,EE235,ENGR111,HUMC201,MATH215
CPE286,CPE312,EE234,EE236,ENGL204,ETHICS,MATH301
AD101,CPE387,EE360,ENGR300,HUMC202,SSCC201
CPE410,CPE408,EE348,FREELEC1,MATH227,SSCC202
CPE471,CPE448,CPE449A,CPE489,EE443,MATH323
CAPS390,CPE449B,ME223,TELEC1.

Each row represents a different semester, the first one being Fall, freshman year, then succeeding spring, fall and so on. While the course dependency rules are met, it was not acceptable to have a course such as MATH227 in the JUNIOR year, such a course should be taken much earlier due to its relative light content and other program specific reasons. More difficult courses that have been scheduled to be taken earlier should be placed instead of MATH227. Because there were no rules that could facilitate such a choice, we introduced the concept of *special requirements*, through which a user can assign a certain requirement cost and so force the algorithm to schedule various courses no later than specified semesters.

In addition, the application did not specify if any of the grouped courses can be swapped or not, in other words, in a semester sequence such as
CPE471,CPE448,CPE449A,CPE489,EE443,MATH323, can a student take CPE410, or CPE460 or CPE473 instead of CPE471?

Another problem was that the application was sometimes outputting hundreds of solutions all in a sequential text file, making it very hard to read and choose for a simple and optimal choice.

At this point the algorithm has been adjusted to solve the above problems, and also optimized again in various parts. We have decided to build a tree of solutions, each semester being a node level, thus converting the relatively discouraging number of solutions into a fairly simple choice, that can easily derive from the student's preference (figure 9).

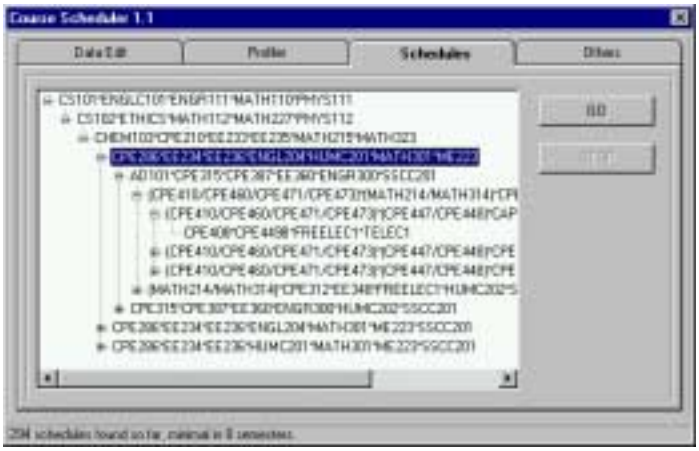

Figure 9. Sample output schedule

The example through figure 12, shows the case of a student that has taken (possibly placement exams suggested) MATH109 and ENGL100, starts in the fall semester, has a restriction of 18 credits per semester, and the following credit limitations: SOPHOMORE, 37, JUNIOR, 71, SENIOR, 105 (Figure 10).

Figure 10. Profiler Scenario

The application was run on a Pentium III 600 computer with 256Mb of RAM, and the output was completed in 73 seconds (a text file containing the solutions in a serial order is also being output).

Although there are almost 300 schedules that would all be acceptable from an advisor's point of view, the student can now easily choose for each semester his / her preferred choice, and continue to expand for the next semesters of his / her choice, while still meeting the constraints of the program and still finishing in the fastest number of semesters possible. Note that there are no possibilities of graduating in less than 8 semesters, and any possibilities that would take longer are being omitted

through the *Maximum Number of Semesters* option from the *Profiler*.

We have also added a way of displaying properly the groups of courses, displaying all of those that can be taken at a given moment, in parenthesis. A progress bar, an option to cancel and few other small features proved as well very useful.

In the example from figure 13, the application found unique semester choices for Fall Freshman, Spring Freshman and Fall Sophomore, after which there are choices.

For the Spring Sophomore semester, the student can either go for **CPE286, EE234, EE236, ENGL204, HUMC201, MATH301, ME223** or **CPE286, EE234, EE236, ENGL204, MATH301, ME223, SSCC201** or **CPE286, EE234, EE236, HUMC201, MATH301, ME223, SSCC201**.

Suppose the student prefers the first choice, he / she can choose for his / her Fall Junior semester either **AD101, CPE315, CPE387, EE360, ENGR300, SSCC201** or **CPE315, CPE387, EE360, ENGR300, HUMC202, SSCC201**.

Suppose the student prefers again the first choice, his / her options for the Fall Junior semester are **(CPE410/CPE460/CPE471/CPE473), (MATH214/MATH314), CPE312, EE348, HUMC202, SSCC202** or **(MATH214/MATH314), CPE312, EE348, FREELEC1, HUMC202, SSCC202.**

Note that the student needs to take only one of the courses from each parenthesis. He / She does not need to check whether he /she qualifies or not for any of them or whether they are offered or not, this being taken care of through the algorithm, Suppose the student prefers the first choice, for his / her Fall Senior year he / she can choose from **(CPE410/CPE460/CPE471/CPE473), (CPE447/CPE448), CAPS390, CPE449A, CPE489, EE443** or **(CPE410/CPE460/CPE471/CPE473), (CPE447/CPE448), CPE449A, CPE489, EE443, FREELEC1** or **(CPE410/CPE460/CPE471/CPE473), (CPE447/CPE448), CPE449A, CPE489, EE443, TELEC1.**

Taking the first choice, the only option for the last semester remains **CPE408, CPE449B, FREELEC1, TELEC1.**

From this point on, an exhaustive sequence of testing and possible scenarios have been circulated through the application by university advisors. Various minor problems have been fixed and we have finally decided on the exact variables and categories that a user need to manipulate. The current Data Manager, Profiler, Schedules and Others has been adopted [7], with all the previously presented features. The output solutions are matching closely to the one proposed by the department, however, the application can find surprisingly more and better solutions, that show their necessity especially when dealing with transfer credits or difficult to meet student preferences.

4.1 A Second Example. Typical Scenario

To demonstrate better the advantages of the application, a second example is given, this time not the case of applicants that have just been admitted as freshmen, but of a transfer student. Student *X* has just been admitted at the University of Bridgeport. He / she has already attended 4 semesters at another university and based on the transfer information, the following list of courses taken are considered taken already: CHEM103, CPE210, CPE286, CPE315, CPE410, CPE460, CS101, CS102, EE235, EE360, ENGL100, ENGLC101, ENGR111, ENGR300, HUMC201, MATH109, MATH110, MATH112, MATH227, MATH301, MATH314, MATH323, PHYS111, PHYS112, SSCC201.

The program requirements at the previous university were different from those of the University of Bridgeport, this making it even more difficult in sorting out what and when can be taken. Finally, the student would like to start in the Spring and in addition, he would hope this time to see few alternatives, as to balance his / her time load with his / her part time job.

After the *Profiler* is being adjusted accordingly (Figure 11),

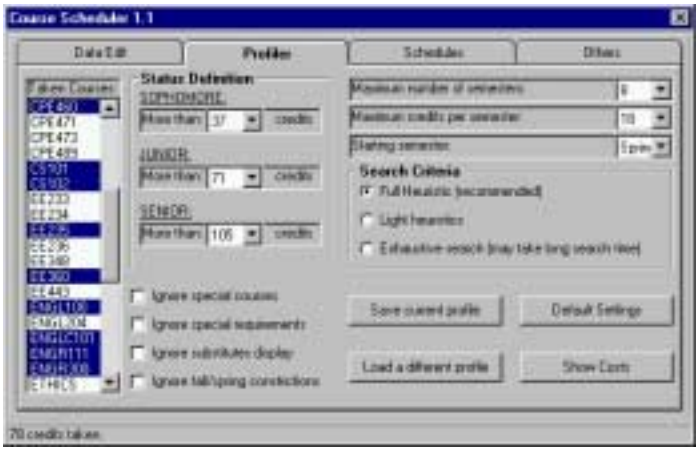

Figure 11. Secondary example profiler

The application outputs a total of 69 different possibilities that could all get the student graduated in 4 semesters (Figure 12).

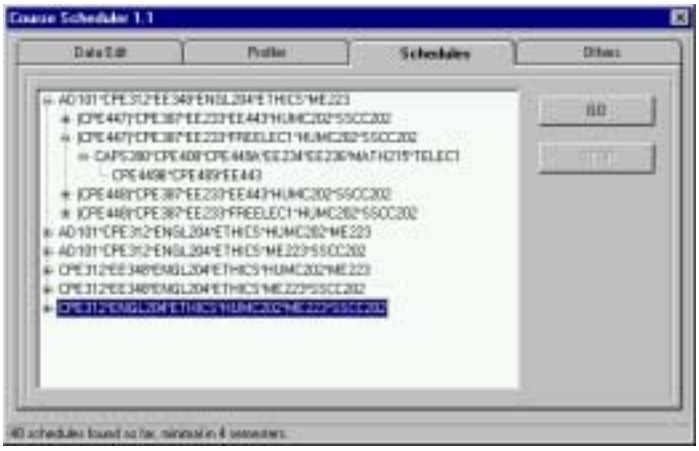

Figure 12. Second example scheduling. (A)

After going over few of the offered choices, the student decides on a schedule that matches closely to his / her needs. He / She will start by taking

AD101, CPE312, EE348, ENGL204, ETHICS, ME223

In the Spring, then continue with

(CPE447), CPE387, EE233, FREELEC1, HUMC202, SSCC202

in the Fall (note the parenthesis for CPE447 denoting one of the group courses, but the fact that it is the only choice at the moment). Next Spring, he / she will take

CAPS390, CPE408, CPE449A, EE234, EE236, MATH215, TELEC1, and then finish next Fall with CPE449B, CPE489, EE443.

However, things do not go exactly as planned for student X. After the first semester, he / she fails course CPE312, and returns for a new advising solution. Although he / she would

have normally graduated in the next 3 semesters, this can not happen anymore due to the fact that CPE312 is a requirement course with a high cost (that is needed by many of the next courses in order to continue). The algorithm outputs 12 possible solutions in a fastest time of 4 semesters (Figure 13).

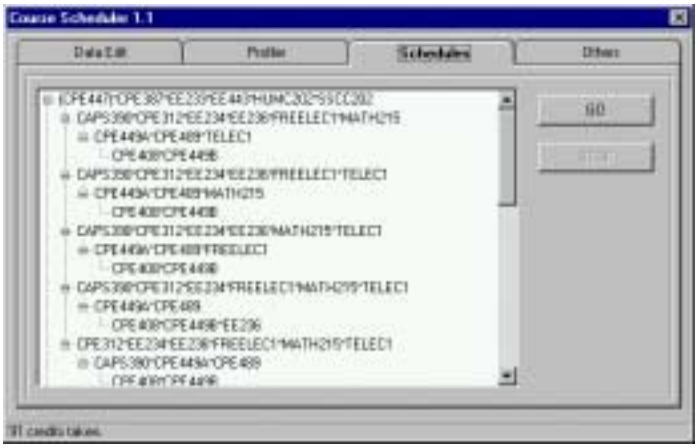

Figure 13. Second example scheduling.(B)

If he / she had failed AD101, EE348 or ME223, he / she would have still been able to graduate in 3 semesters. A quick look at the remaining courses reveals this clarity of the problem. For example, a student can only take CPE449B after he / she took CPE449A. CPE449A has as requirements CPE312, ETHICS and ENGL204, and for example CPE312 is not being offered in fall, which brings up to an obvious minimal of 4 semesters. Given the situation, the student chooses for the fall CPE447, CPE387, EE233, EE443, HUMC202, SSCC202, then he / she plans CAPS390, CPE312, EE234, FREELEC1, MATH215 and TELEC1 for the spring, CPE449A and CPE489 for the next fall and finally CPE449B, CPE408 and EE236.
Although in the next semester the student fails course EE443, a new rescheduling shows that he / she can still graduate in three more semesters (Figure 14).

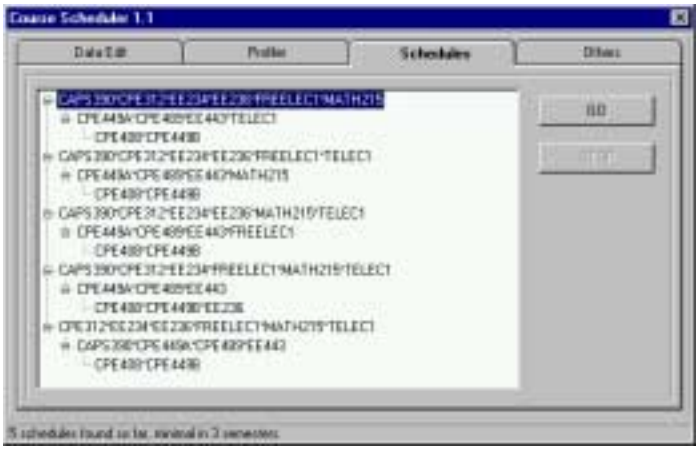

Figure 14. Second example scheduling. (C)

The student will continue with his / her previously selected schedule, but now will add EE443 to his / her first senior semester.

4.2 A Third Example. Specific Scenario A

Over the next three examples, the emphasis will go on exposing the usefulness of the application in rather unusual or tedious (time consuming) situations.
Student *X* arrives as a transfer student at University of Bridgeport, and the school from where he transferred simply had no course dependency requirements in their scheduling, allowing for the student's arbitrary schedule at their own risk. After evaluating his / her transcript, this reveals that he has taken the following courses so far:

CAPS390, CHEM103, CPE210, CPE286, CPE312, CPE315, CPE387, CPE408, CPE410, CPE447, CPE448, CPE449A, CPE449B, CPE460, CPE471, CPE473, CPE489, CS101, CS102, EE233, EE234, EE235, EE236, EE348, EE360, EE443, ENGL100, ENGL204, ENGLC101, ENGR111, ENGR300, ETHICS, FREELEC1, HUMC201, HUMC202, MATH109, MATH110, MATH112, MATH214, MATH215, MATH227, MATH301, MATH314.

Overall, 124 credits have been taken. While few courses are left to be taken, which could suggest a limited amount of choices and therefore little time spent in solving them, the work becomes tedious when mapping the course dependency imposed at University of Bridgeport. Making a mistake becomes a highly probable case which could result in affecting the quality of the course curricula at the university. By using the scheduling tool, this is a trivial task after which will reveal in seconds a correct solution such as the following:

- first semester: AD101, ME223, PHYS111, SSCC201, TELEC1
- second (last) semester: MATH323, PHYS112, SSCC202

Also to consider is the fact that the student can actually ask for explanations on why choose a certain schedule, task that can be easily tackled with the aid of the package as well.

4.3 A Fourth Example. Specific Scenario B

This time, let us consider the case of transfer students that have taken very few credits and again, not necessarily adhering to course dependencies. Student *X* has taken 16 credits so far as follows:
CAPS390, CPE387, CPE408, ENGL101, MATH109.
While trying to setup a course schedule, the advisor will notice a hard time in finding a schedule in less than 8 semesters, which indeed is not possible in this case. The amount of work necessary to reach this conclusion can be frustrating and still leave doubts among the student and the advisor, also leaving the desire of a software package that could double check these results. Trying this scenario through this software package will inspire confidence plus a high choice of possibilities.

4.4 A Fifth Example. Specific Scenario C

And finally a last unusual scenario adding to the critical advantage of this software package.
Student *X* is transferring from a university that does respect a course dependency fairly close to the one of the University of

Bridgeport, but is also more flexible on the number of credits per semester. As a result, student *X* has taken a total of 24 credits in one semester, as follows:
CS101, CPE210, ENGL100, ENGR111, MATH109, MATH110 and PHYS111
Due to personal reasons, the student can only attend at University of Bridgeport if he can be accommodated to graduate in 6 semesters (e.g. the case of an international student that comes to study to the United States and has a time limited visa) Trying to come up with a 6 semester solution in regular condition will not succeed, plus the task to justify and determine this will take time. The idea then would be to try to reduce some of the restrictions: allow more credits per semester? How many? Each case will require a thorough attention for an advisor and can be hard to follow. The goal is to just find a solution, and if a tool could allow to "play" with few parameters and come up with something, the issue would be set.
By inputting the scenario in the software package, it appears that the student can actually graduate in 6 semesters if allowed to take up to 22 courses per semester and would start in Spring, bellow listed one of the 8 such possible solutions:
- *first semester*: CHEM103, CS102, ENGLC101, MATH112, MATH227, PHYS112
- *second semester*: CPE315, EE233, EE235, HUMC201, MATH215, MATH323, SSCC201
- *third semester*: (MATH314), CPE286, CPE312, EE234, EE236, ENGL204, ETHICS, MATH301, ME223
- *fourth semester*: (CPE410/CPE460), (CPE447), AD101, CPE387, EE360, ENGR300, HUMC202, SSCC202
- *fifth semester*: (CPE410/CPE460/CPE471), CAPS390, CPE408, CPE449A, EE348, FREELEC1, TELEC1
- *and sixth semester*: CPE449B, CPE489, EE443

5. Current Limitations and Future Work

Although in its current stage the application has enough features to be conveniently used for advising, there are still features that need attention. Students should be able to filter the list of final solutions for their own preference. The *availability cost* of a course should also take in consideration the total number of students a course can be offered to, and to co-relate this fact with a global database that keeps track whether a course is still available from this point of view. Some courses could also happen to be offered in the same exact time, this being a case that the algorithm should consider as well.
Besides the relatively immediate changes above, the software will be probably converted to a completely web based interface too, which would link and maintain a school database, with all the courses for all the majors and all the information for a student stored there as well.
The ultimate goal is to have to student type an ID number from his / her home computer and no other additional information. Based on the id, the software will find the *major* the student belongs too and what courses did he /she take already. Consequently, will output immediately the optimal possibilities and allow him to choose among them and register online. Once the student confirms the selection, this reply gets appended to his / her advisor's profile for double checking (to avoid any possible application error or invalid result), and if the advisor agrees with it, the student will be notified by email and the registration process is complete.

6. Conclusions

In this paper, we present a software model designed to aid the students and the advisors with the tedious and time consuming registration tasks that every semester, every student and advisor need to go through.
The designed application can virtually eliminate the time an advisor would spend with a student on registration, optimizing for the quickest graduation and facilitating student's preferences, thus allowing time for more specific and important student related issues.
Currently we have successfully tested and used the application on the Bachelors of Computer Engineering Major at the University of Bridgeport. The tool provided excellent results. We are also in the process of completely revamping the application to the specifications detailed in Section 5, which would result in a completely automated advising and registration system complying with the requirements of a program of study.

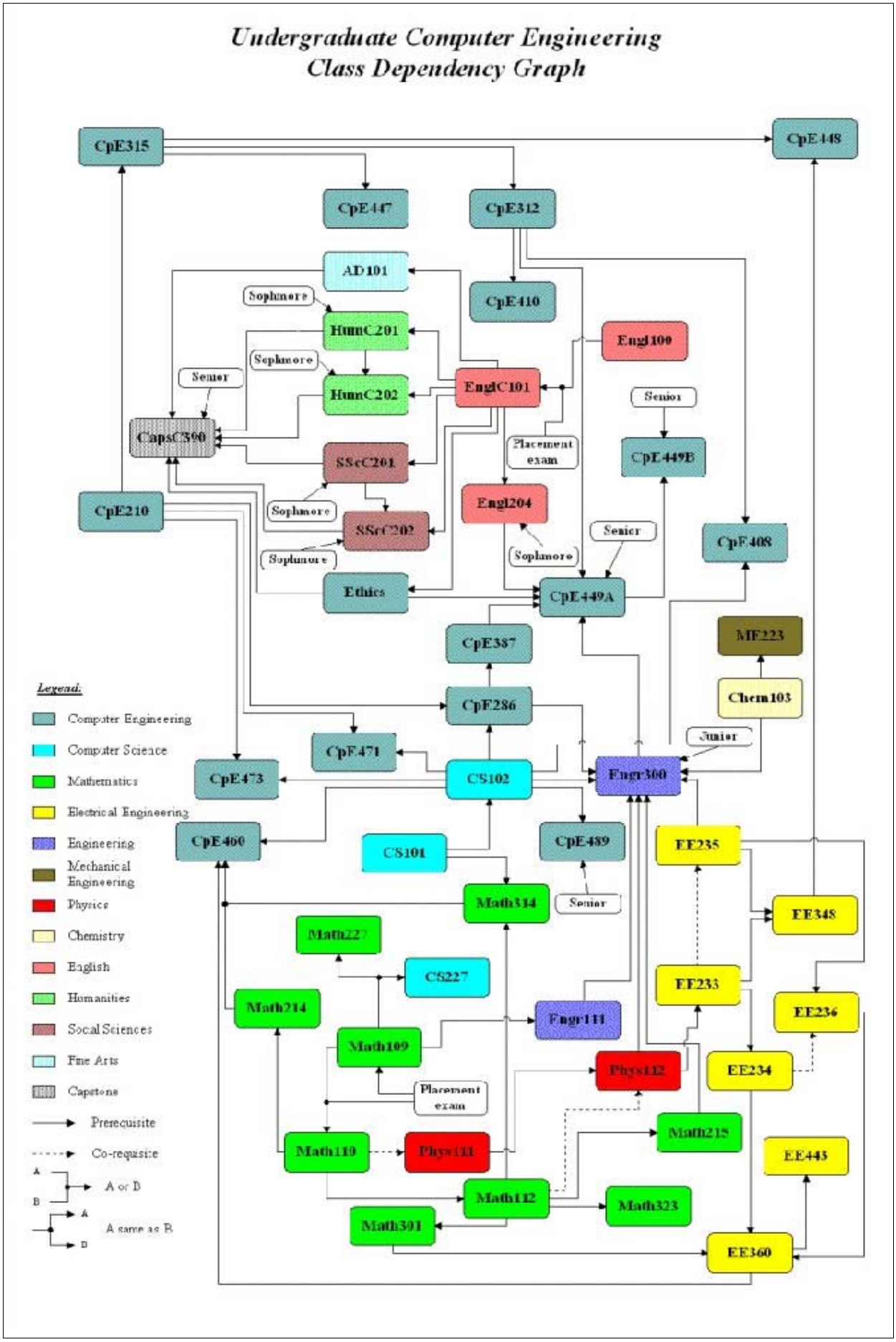

Figure 15.Course Dependency Graph categorized by types of courses.

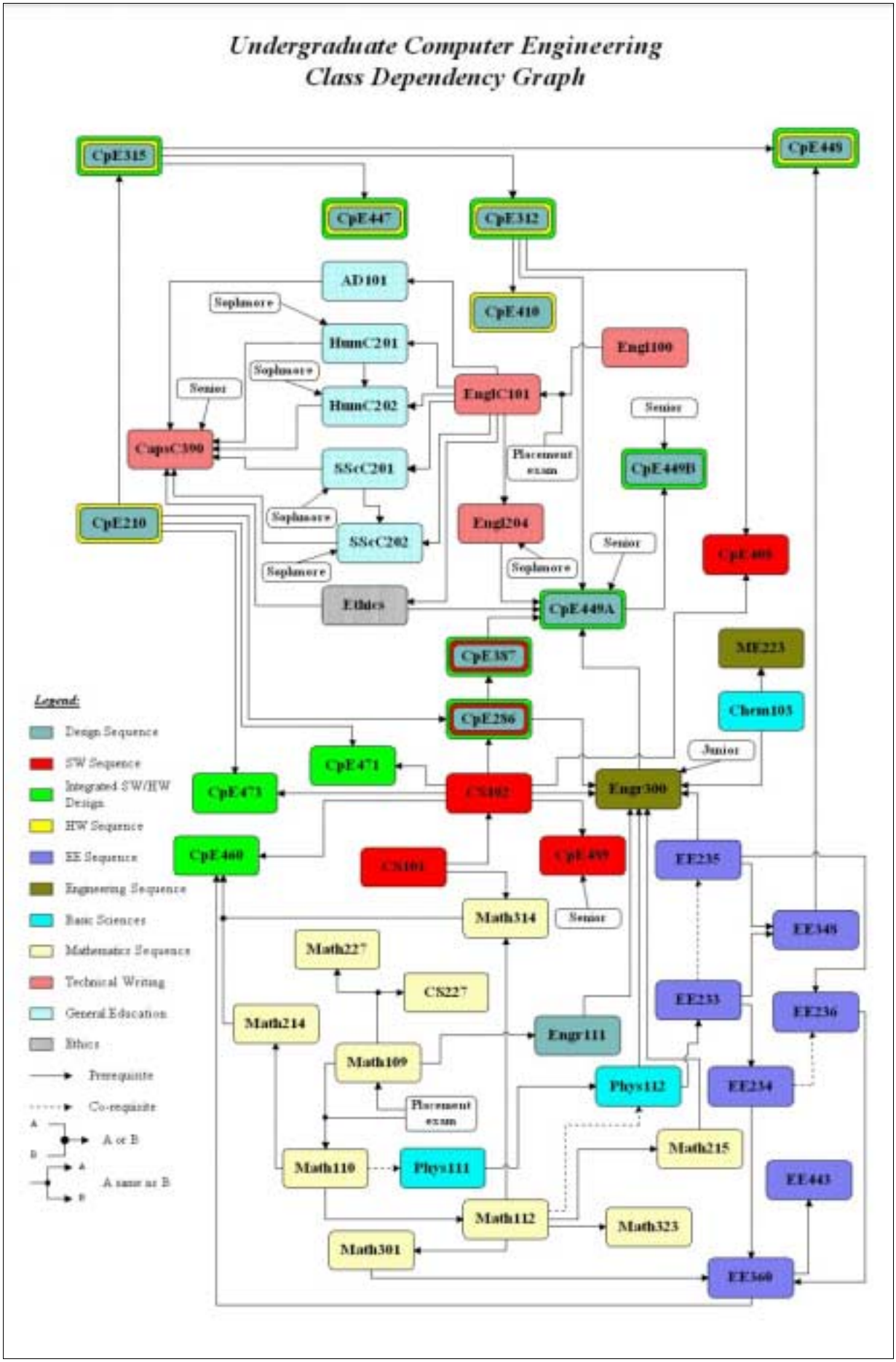

Figure 16.Course Dependency Graph categorized by course sequences.

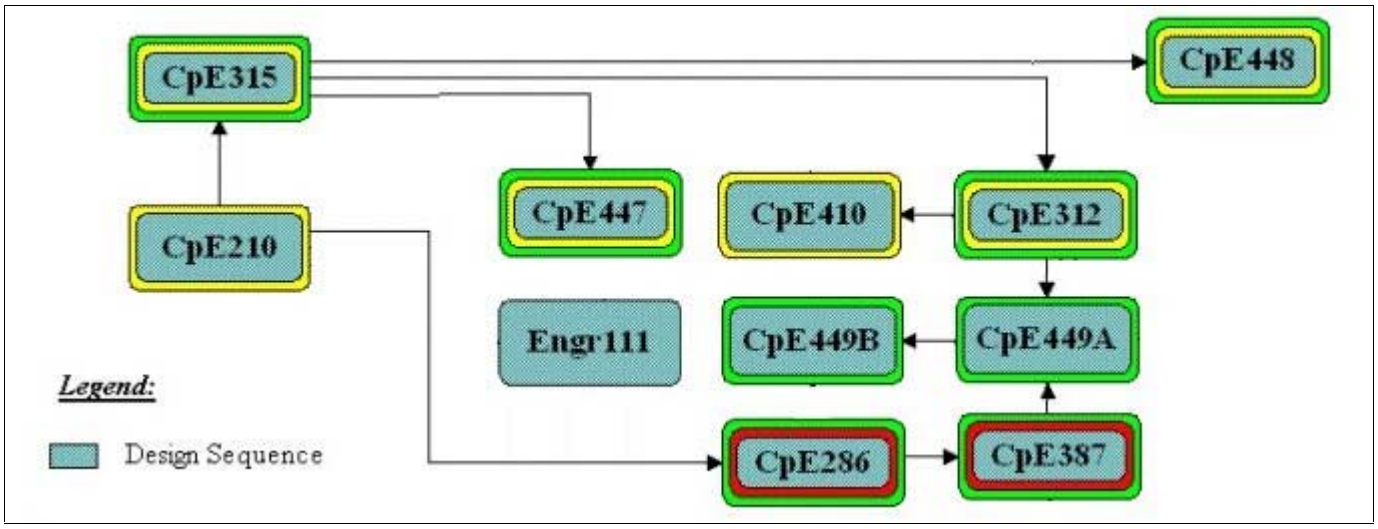

Figure 17.Design Sequence

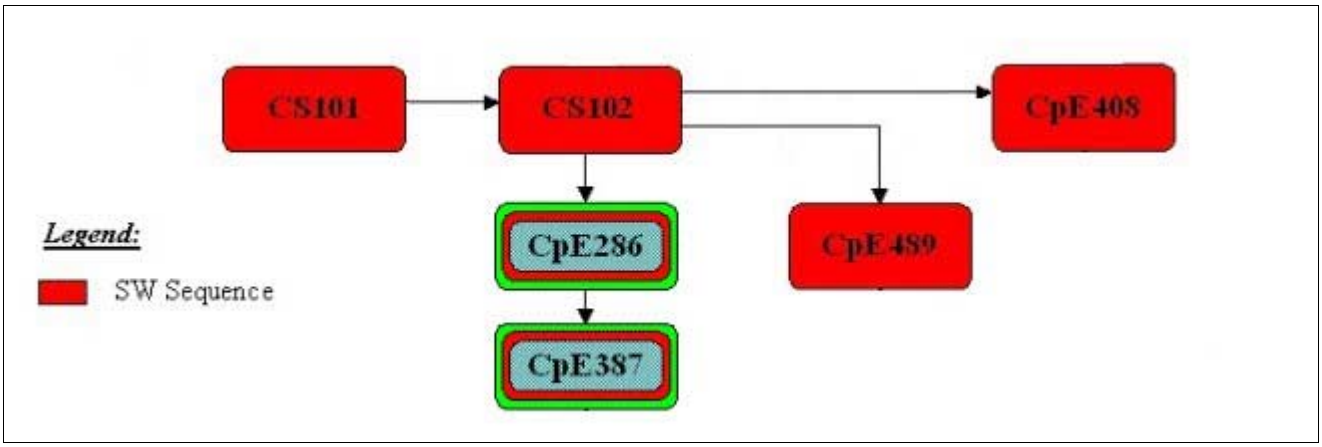

Figure 18.Software Sequence.

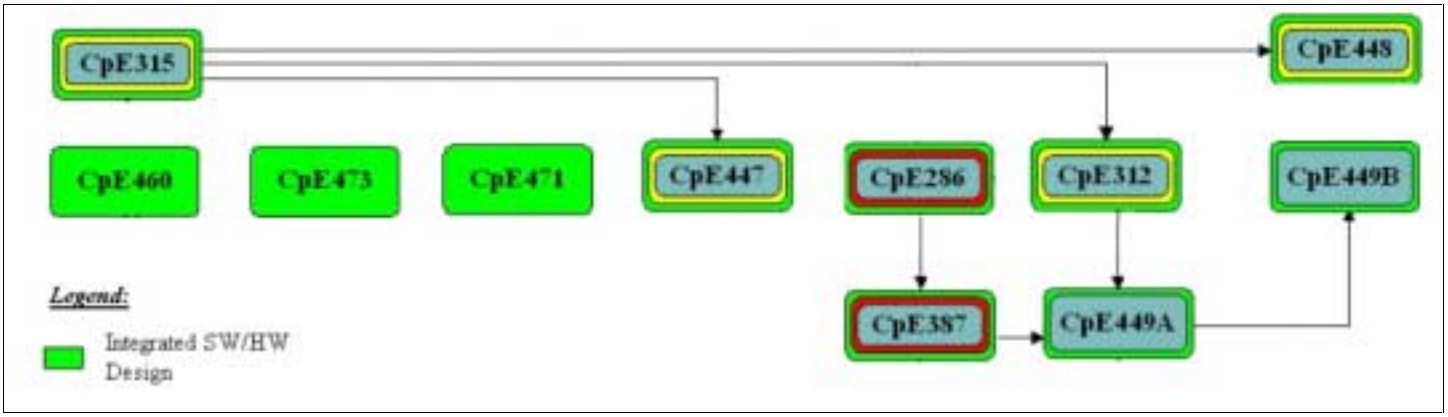

Figure 19.Integrated SW/HW Design Sequence.

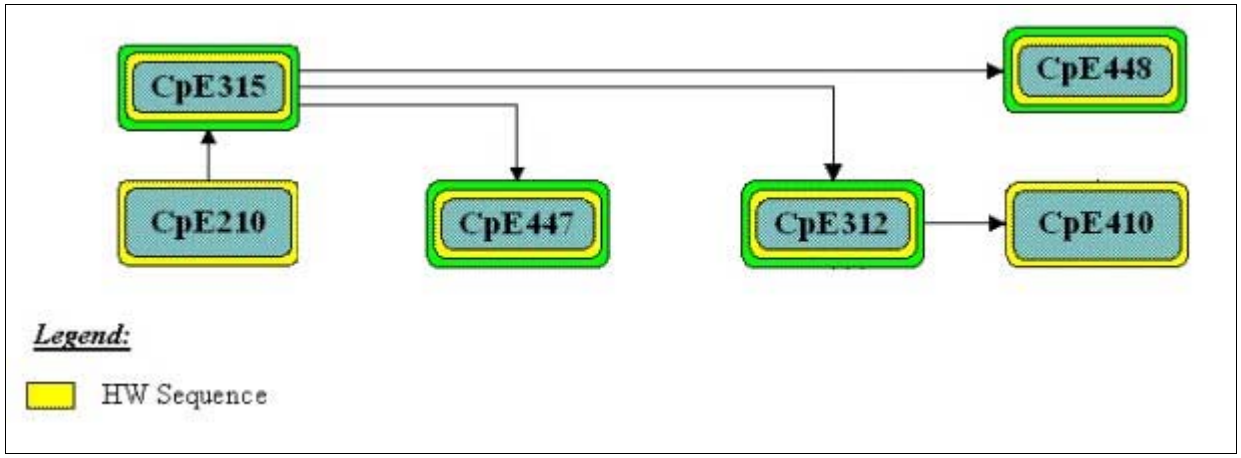

Figure 20. HW Sequence.

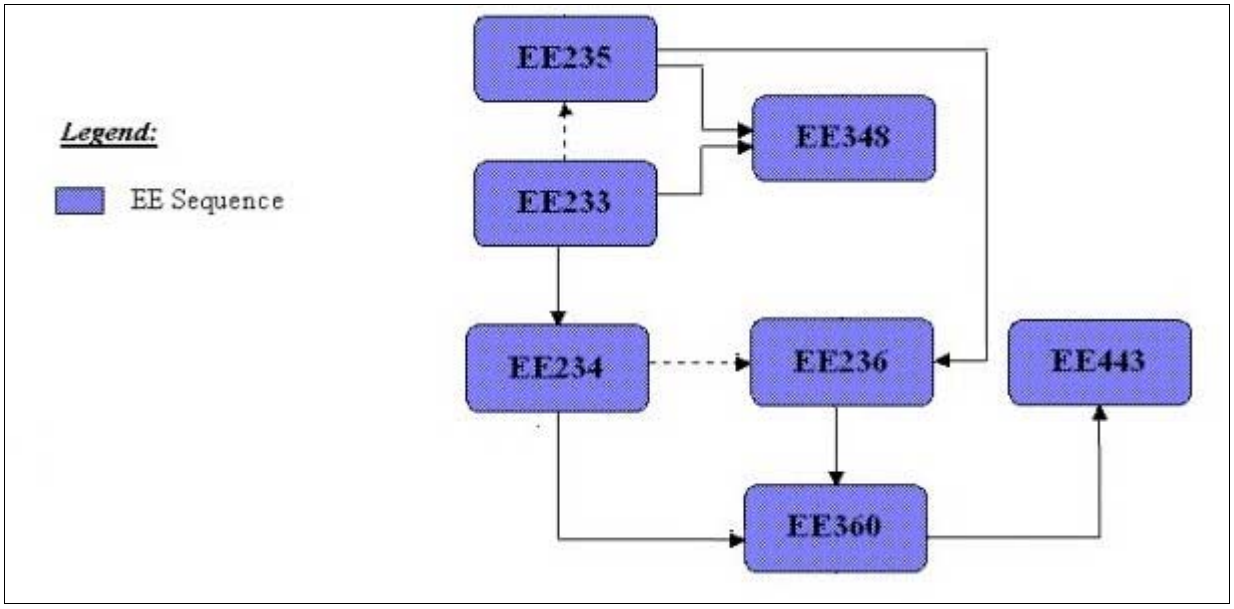

Figure 21. EE Sequence.

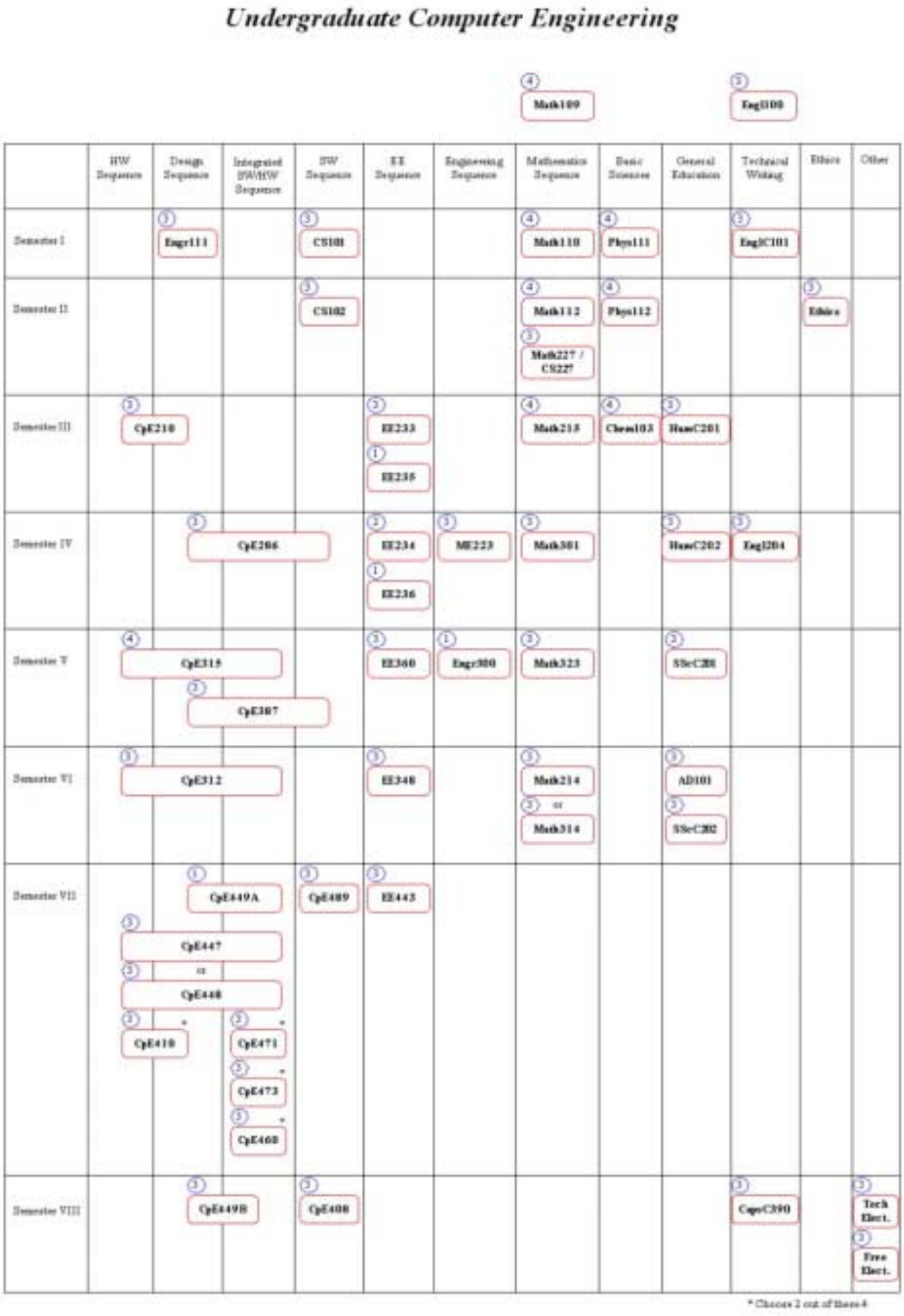

Figure 22. A suggested Schedule

References

[1] Aarts, E. H. L. , Lenstra, Jan Karel, Aarts, Emile L., *"Local Search in Combinatorial Optimization"*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 1997

[2] Greene, Daniel H., Knuth, Donald E., *"Mathematics for The Analysis o Algorithms", Third Edition*, Birkhauser, 1990

[3] Dasgupta, Pallab, Chakrabarti, P. P., Desarkar, S. C., *"Multiobjective Heuristic Search: An Introduction to Intelligent Search Methods for Multicriteria Optimization*Kaufmann Publishers; , 1999

[4] Patrascoiu, Octavian, Marian, Gheorghe, Mitroi, Nicolae, *"Elements of Graphs and Combinatorial Theory, Methods, Algorithms and Programs"*, B.I.C. All, Romania, 1994

[5] Izvercian, P.N., Cretu, V., Izvercian, M., Resiga, R., *"Introduction in Graph Theory, The Critical Path Method"*, Editura de Vest, Romania, 1993

[6] Graham, Ronald L., Knuth, Donald E., Patashnik, Oren, *"Concrete Mathematics, A Foundation for Computer Science"*, Addison-Wesley, 1994
Control. Second Edition",1989, Addison Wesley

[7] Markotty, Michael, *"Software Implementation (Practical Software Engineering, Vol 4)"*, Prentice-Hall ECS Professional, 1991

Appendix

AD101, Fine Arts, (3 credits)
CAPS390, Capstone Seminar, (3 credits)
CHEM103, General Chemistry I, (4 credits)
CPE210, Digital Design I, (3 credits)
CPE286, Introduction to Microprocessors, (3 credits)
CPE312, Computer Organization, (3 credits)
CPE315, Digital Design II with Laboratory, (4 credits)
CPE387, Embedded System Design, (3 credits)
CPE408, Operating Systems, (3 credits)
CPE410, Introduction to Computer Architecture, (3 credits)
CPE447, FPGA Design, (3 credits)
CPE448, Introduction to VLSI Design, (3 credits)
CPE449A, Senior Project part A, (1 credits)
CPE449B, Senior Project part B, (3 credits)
CPE460, Introduction to Robotics, (3 credits)
CPE471, Computer Communications I: System Analysis, (3 credits)
CPE473, Local Area Networks, (3 credits)
CPE489, Software Engineering, (3 credits)
CS101, Introduction to Computing I, (3 credits)
CS102, Introduction to Computing II, (3 credits)
EE233, Network Analysis I, (3 credits)
EE234, Network Analysis II, (2 credits)
EE235, Network Analysis I Lab, (1 credits)
EE236, Network Analysis II Lab, (1 credits)
EE348, Electronic Circuits I, (3 credits)
EE360, Controls, (3 credits)
EE443, Applied Digital Signal Processing, (3 credits)
ENGL100, Basic Composition, (3 credits)
ENGL204, Technical Writting for Comp. Sci. & Eng., (1 credits)
ENGLC101, Composition and Rhetoric I, (3 credits)
ENGR111, Introduction to Engineering I, (3 credits)
ENGR300, Economics and Management of Engineering Projects, (1 credits)
ETHICS, Integrated Studies in Computing (INTSC101), (3 credits)
FREELEC1, Free Elective 1, (3 credits)
HUMC201, Introduction to Humanities I, (3 credits)
HUMC202, Introduction to Humanities II, (3 credits)
MATH109, Precalculus Mathematics, (4 credits)
MATH110, Calculus and Analytic Geometry I, (4 credits)
MATH112, Calculus and Analytic Geometry II, (4 credits)
MATH214, Linear Algebra, (3 credits)
MATH215, Calculus and Analytic Geometry III, (4 credits)
MATH227, Discrete Structures, (3 credits)
MATH301, Differential Equations, (3 credits)
MATH314, Numerical Methods, (3 credits)
MATH323, Probability and Statistics, (3 credits)
ME223, Materials Science for Engineers, (3 credits)
PHYS111, Principles of Physics I, (4 credits)
PHYS112, Principles of Physics II, (4 credits)
SSCC201, Introduction to the Social Sciences I, (3 credits)
SSCC202, Introduction to the Social Sciences II, (3 credits)
TELEC1, Technical Elective 1, (3 credits)